Relatório do 1º projeto de Algoritmo e Estrutura de Dados (AED)

Projeto: **Comparação de Métodos de Resolução do Subset-Sum Problem**

Grupo: Guilherme Dias (nº 103128), 50%

Tomás Almeida (nº 103300), 50%

Data de entrega: 9 de Janeiro 2021

Docentes: Tomás Oliveira e Silva (TP) e Pedro Lavrador (P)

Índice

Introdução 3
Explicação dos Métodos 4
Código Brute Force.....….5
Código Clever Brute Force.....….7
Código Meet in the Middle.....….9
Gráficos Comparação entre os diferentes métodos..... 14
Código Programa em Matlab.....….15
Conclusão…….....17
Bibliografia..... 17

Introdução

Este trabalho tem como objetivo encontrar diferentes soluções para o Subset-Sum Problem, ou seja, soluções para encontrar de entre um conjunto de números, quais destes somados dão um outro número anteriormente definido.
Implementámos então 3 algoritmos diferentes para resolver o problema, e utilizámos os dados obtidos de cada resolução para desenhar um gráfico que permite comparar o desempenho das 3 soluções.

Explicação dos métodos

Para cada problema temos três componentes: o conjunto de números que serão somados; o número para o qual procuramos as somas dentro do conjunto; o binário que define que números foram e não foram utilizados na soma. O objetivo final das três implementações é o mesmo: encontrar a o conjunto de números que somado resulta no valor procurado. Ou seja, o resultado para cada problema é apresentado através de 1's e 0's, na qual cada número representa um elemento do conjunto. Caso o elemento esteja representado por um 0, este não pertence à soma final, caso esteja representado por um 1, significa que é um dos números usado para chegar à soma.

No primeiro algoritmo (Brute Force), são realizadas todas as somas possíveis do conjunto dos números, até eventualmente encontrar aquela que tem como resultado o valor pretendido. Este algoritmo é claramente o menos eficiente, pois pode vir a ter que percorrer todos os subconjuntos possíveis no pior dos casos, e é também aquele com maior complexidade computacional (O(2^n)).

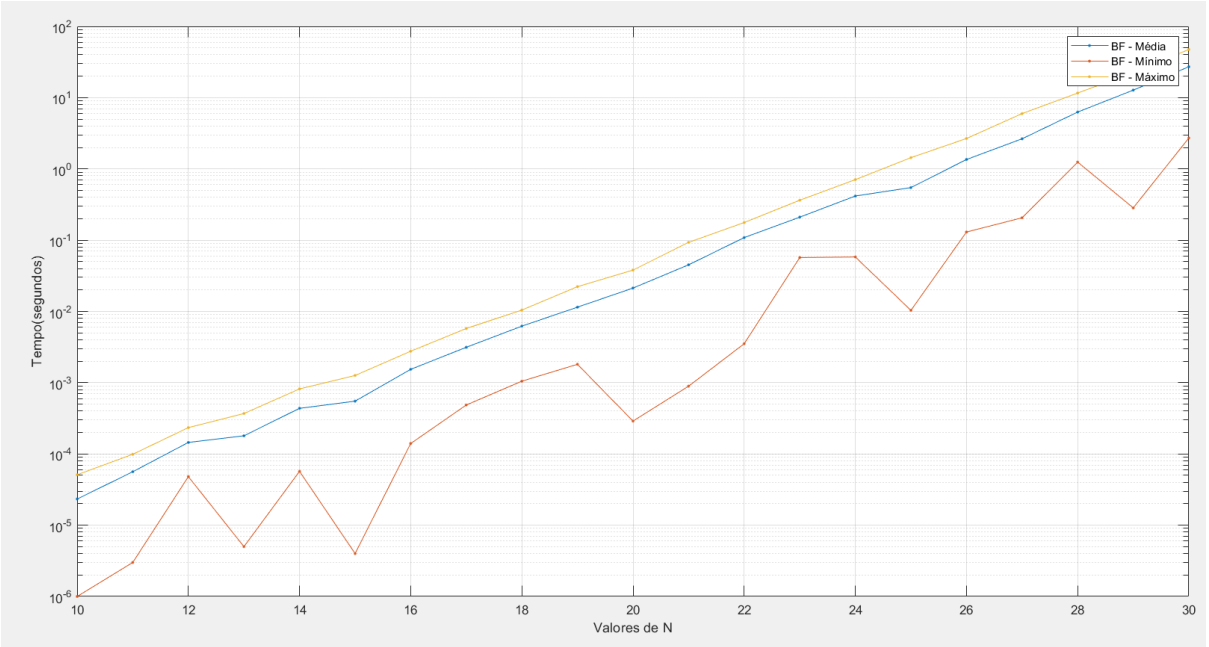
No segundo algoritmo (Brute Force Inteligente), é usada uma função recursiva que evita novas recursões quando é verificado que com os dados obtidos, nenhuma solução é possível pois a soma parcial é muito pequena ou muito grande. Desta forma a complexidade computacional diminui bastante relativamente ao 1º método, e consequentemente o tempo de execução também diminui.

No terceiro algoritmo (Meet in the Middle), divide-se o conjunto inicial em dois arrays, e são realizados todos os subconjuntos possíveis desses 2 conjuntos (sub sums). Ordenamos os dois subconjuntos, e realizamos a soma do primeiro elemento do primeiro conjunto e do último elemento do segundo conjunto, se o resultado for maior do que o pretendido, descemos 1 no index do segundo conjunto e voltamos a somar, se o resultado for menor que o pretendido, o index do primeiro conjunto aumenta 1 e voltamos então a realizar a soma. Este método é de longe o mais eficiente, o mais rápido, e o que tem menor complexidade computacional (O(2^n/2)).

Código Brute Force

```
//Função Brute-force
char bruteForce(int n,integer_t p[],integer_t desired_sum,int result[])
{
 for (int comb = 0;comb<(1<<n); comb++){

    integer_t test_sum=0;

    for (int bit = 0; bit < n; bit++){
        if (comb & (1<<bit)){
            test_sum += p[bit];
            result[bit]=1;
        }

        else{
            result[bit]=0;
        }
  }

    if (test_sum == desired_sum)
        return 1;


 }

 return 0; //valor desired_sum não encontrado
}
```

```
found = bruteForce(n, p, desired_sum, result);
```



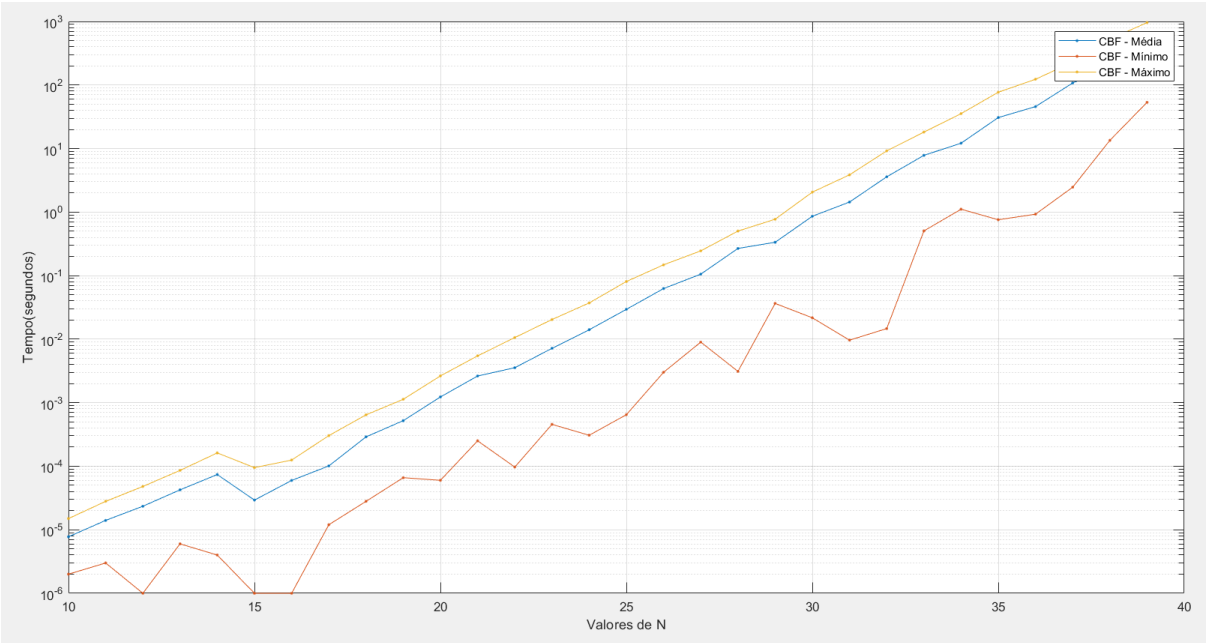

```
Para n = 23 | Found: 1 | Time: 0.255541 seconds Result: 00011010101101100000101
Para n = 23 | Found: 1 | Time: 0.372544 seconds Result: 01101100100101110010111
Para n = 23 | Found: 1 | Time: 0.131266 seconds Result: 10010101111010001001010
Para n = 23 | Found: 1 | Time: 0.203445 seconds Result: 00011011111000001000001
Para n = 23 | Found: 1 | Time: 0.147904 seconds Result: 01001011011011110111010
Para n = 23 | Found: 1 | Time: 0.302769 seconds Result: 01000001010111101111101
Para n = 24 | Found: 1 | Time: 0.597726 seconds Result: 001011101011110111101101
Para n = 24 | Found: 1 | Time: 0.801288 seconds Result: 100100110111000110101111
Para n = 24 | Found: 1 | Time: 0.039347 seconds Result: 111100100001000000110000
Para n = 24 | Found: 1 | Time: 0.297309 seconds Result: 001110010100110000111010
Para n = 24 | Found: 1 | Time: 0.151765 seconds Result: 010110100001010001110100
Para n = 24 | Found: 1 | Time: 0.274769 seconds Result: 110101110000110101101010
Para n = 24 | Found: 1 | Time: 0.368446 seconds Result: 010001010010001111001110
Para n = 24 | Found: 1 | Time: 0.077204 seconds Result: 111100101000111011101000
Para n = 24 | Found: 1 | Time: 0.052474 seconds Result: 100111101101101111110000
Para n = 24 | Found: 1 | Time: 0.770004 seconds Result: 111010001000111111110111
Para n = 24 | Found: 1 | Time: 0.473196 seconds Result: 101111111100011100101001
Para n = 24 | Found: 1 | Time: 0.328281 seconds Result: 010010111101101101100110
Para n = 24 | Found: 1 | Time: 0.401845 seconds Result: 010011101111101001111110
Para n = 24 | Found: 1 | Time: 0.647486 seconds Result: 111111111100000110010011
Para n = 24 | Found: 1 | Time: 0.484538 seconds Result: 010001111010001111101001
Para n = 24 | Found: 1 | Time: 0.753992 seconds Result: 110010100010110101010111
Para n = 24 | Found: 1 | Time: 0.266697 seconds Result: 100101000110011101001010
Para n = 24 | Found: 1 | Time: 0.094529 seconds Result: 101001010011110100111000
Para n = 24 | Found: 1 | Time: 0.566875 seconds Result: 111010011000110110001101
Para n = 24 | Found: 1 | Time: 0.531672 seconds Result: 111101100000100001100101
Para n = 25 | Found: 1 | Time: 0.070129 seconds Result: 0101010100000100101010000
Para n = 25 | Found: 1 | Time: 0.202934 seconds Result: 0011011100100000011111000
Para n = 25 | Found: 1 | Time: 1.320296 seconds Result: 1101110001000001111010011
Para n = 25 | Found: 1 | Time: 0.106026 seconds Result: 1000001000010010000001000
Para n = 25 | Found: 1 | Time: 0.532339 seconds Result: 1110101101100100101001010
Para n = 25 | Found: 1 | Time: 0.220026 seconds Result: 1111101011010000110000100
Para n = 25 | Found: 1 | Time: 0.528894 seconds Result: 1100011000011011110001010
Para n = 25 | Found: 1 | Time: 1.602890 seconds Result: 1001001110111111110101111
Para n = 25 | Found: 1 | Time: 0.273227 seconds Result: 0010110010000010001010100
Para n = 25 | Found: 1 | Time: 0.420976 seconds Result: 1101010111010111000000010
Para n = 25 | Found: 1 | Time: 0.415463 seconds Result: 1011011010100110010111100
Para n = 25 | Found: 1 | Time: 0.636102 seconds Result: 0011010100101001100000110
Para n = 25 | Found: 1 | Time: 1.129392 seconds Result: 1111110111100010101110101
Para n = 25 | Found: 1 | Time: 0.993797 seconds Result: 1110110101101101011101001
Para n = 25 | Found: 1 | Time: 1.550005 seconds Result: 0001001010000010111100111
Para n = 25 | Found: 1 | Time: 0.500013 seconds Result: 1000011110110000001010010
Para n = 25 | Found: 1 | Time: 0.062600 seconds Result: 0101010100110110010010000
Para n = 25 | Found: 1 | Time: 0.617097 seconds Result: 0010010010110101111011010
Para n = 25 | Found: 1 | Time: 1.583686 seconds Result: 1101011011000001010010111
Para n = 25 | Found: 1 | Time: 0.485901 seconds Result: 0101110111001001001010010
Para n = 26 | Found: 1 | Time: 0.078554 seconds Result: 00100000000110001110100000
Para n = 26 | Found: 1 | Time: 1.715745 seconds Result: 11111101110111000011000001
```

Código Clever Brute Force

```
char branchAndBound(int n,integer_t *p,integer_t desired_sum,integer_t current_sum,int current_index,int result[])
{
    if(current_sum==desired_sum)
        return 1;

    if(current_sum>desired_sum)
        return 0;

    if(current_index==n)
        return 0;


    for (int index = current_index; index < n; index++){
        result[index] = 1;
        if(branchAndBound(n, p, desired_sum, current_sum + p[index], ++current_index,result))
            return 1;
        result[index]= 0;
    }

    return 0;
}
```

```
found = branchAndBound(n, p, desired_sum, 0, 0, result);
```

```
Para n = 29 | Found: 1 | Time: 0.096742 seconds Result: 11010110010010000000011000111
Para n = 29 | Found: 1 | Time: 0.775268 seconds Result: 01100001100111010010111110000
Para n = 29 | Found: 1 | Time: 0.264235 seconds Result: 11000111100101111110010100100
Para n = 29 | Found: 1 | Time: 0.070391 seconds Result: 11011000000111010000010101010
Para n = 29 | Found: 1 | Time: 0.731051 seconds Result: 01100011101110111101110000001
Para n = 29 | Found: 1 | Time: 0.021460 seconds Result: 11111100001110110110101110011
Para n = 29 | Found: 1 | Time: 0.829945 seconds Result: 01001011011010101010001101001
Para n = 29 | Found: 1 | Time: 0.063916 seconds Result: 11001000010100110010110000100
Para n = 29 | Found: 1 | Time: 1.048938 seconds Result: 00011111011000110110100100110
Para n = 29 | Found: 1 | Time: 1.234960 seconds Result: 00000110000110011011101110001
Para n = 29 | Found: 1 | Time: 0.547615 seconds Result: 01011100101100111100011000001
Para n = 30 | Found: 1 | Time: 0.266617 seconds Result: 01111001010111101000010100000
Para n = 30 | Found: 1 | Time: 1.039539 seconds Result: 00100111111101010110000010100
Para n = 30 | Found: 1 | Time: 0.278564 seconds Result: 10011110100010111011110000000
Para n = 30 | Found: 1 | Time: 0.273936 seconds Result: 01101001000001100001010001001
Para n = 30 | Found: 1 | Time: 0.269753 seconds Result: 11101001101101101110010100111
Para n = 30 | Found: 1 | Time: 2.872360 seconds Result: 00000010110011001111101001010
Para n = 30 | Found: 1 | Time: 0.239544 seconds Result: 10000110111101110000000010010
Para n = 30 | Found: 1 | Time: 1.653528 seconds Result: 01111111111110100111100110100
Para n = 30 | Found: 1 | Time: 1.690212 seconds Result: 01110111000111111110000011001
Para n = 30 | Found: 1 | Time: 0.655592 seconds Result: 01000101010100010010110001100
Para n = 30 | Found: 1 | Time: 0.460118 seconds Result: 11010100011011101100100100110
Para n = 30 | Found: 1 | Time: 0.849686 seconds Result: 01110100001011011000011100100
Para n = 30 | Found: 1 | Time: 1.212916 seconds Result: 10000100111100011110100110100
Para n = 30 | Found: 1 | Time: 2.401134 seconds Result: 01001001000101001100101101111
Para n = 30 | Found: 1 | Time: 1.195549 seconds Result: 10100001111010100111111010010
Para n = 30 | Found: 1 | Time: 0.267170 seconds Result: 01011100000010110110100000001
Para n = 30 | Found: 1 | Time: 0.194705 seconds Result: 10101001011001011000010010000
Para n = 30 | Found: 1 | Time: 0.838825 seconds Result: 10110101011100111110100110010
Para n = 30 | Found: 1 | Time: 1.482801 seconds Result: 01001000001101110010110001010
Para n = 30 | Found: 1 | Time: 1.427681 seconds Result: 00111001100000100011110101010
Para n = 31 | Found: 1 | Time: 3.407482 seconds Result: 01101111111111001110100100010111
Para n = 31 | Found: 1 | Time: 2.887451 seconds Result: 00101111100101100100110010110000
Para n = 31 | Found: 1 | Time: 5.489805 seconds Result: 00001001010000101111000110111010
Para n = 31 | Found: 1 | Time: 1.089879 seconds Result: 00101110111100110000000000110100
Para n = 31 | Found: 1 | Time: 1.562059 seconds Result: 10110101111101101110111001010000
Para n = 31 | Found: 1 | Time: 1.562059 seconds Result: 10100101000111101001011111000000
Para n = 31 | Found: 1 | Time: 2.639751 seconds Result: 01001110001001000101111011100000
Para n = 31 | Found: 1 | Time: 5.183078 seconds Result: 00100101100011110111010000011111
Para n = 31 | Found: 1 | Time: 1.518651 seconds Result: 00111000010100010110010100100001
Para n = 31 | Found: 1 | Time: 4.019388 seconds Result: 01011101110111011111011100011110
Para n = 31 | Found: 1 | Time: 0.880219 seconds Result: 10011000010100000010000011101101
Para n = 31 | Found: 1 | Time: 0.411224 seconds Result: 11101100110010001101011101001010
Para n = 31 | Found: 1 | Time: 3.297330 seconds Result: 01001100001011101111010010110100
Para n = 31 | Found: 1 | Time: 5.255520 seconds Result: 00001001010110000100111001011111
Para n = 31 | Found: 1 | Time: 1.940689 seconds Result: 10001110010110000001011000110111
Para n = 31 | Found: 1 | Time: 1.617626 seconds Result: 00011011100100000011100010110000
Para n = 31 | Found: 1 | Time: 2.722277 seconds Result: 10001111001100000010110110111101
```


CBF - Média
CBF - Mínimo
CBF - Máximo
Tempo(segundos)
Valores de N
10
15
20
25
30
35
40

Código Meet in the Middle

```c
void swap(integer_t *a, integer_t *b)
{
    integer_t t = *a;
    *a = *b;
    *b = t;
}



/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
    array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
integer_t partition (integer_t array[], integer_t low, integer_t high)
{
    // pivot (Element to be placed at right position)
    integer_t pivot = array[high];

    integer_t i = (low - 1);  // Index of smaller element and indicates the
                              // right position of pivot found so far

    for (integer_t j = low; j <= high- 1; j++)
    {
        // If current element is smaller than the pivot
        if (array[j] < pivot)
        {
            i++;    // increment index of smaller element
            swap(&array[i], &array[j]);
        }
    }
    swap(&array[i + 1], &array[high]);
    return (i + 1);
}
```

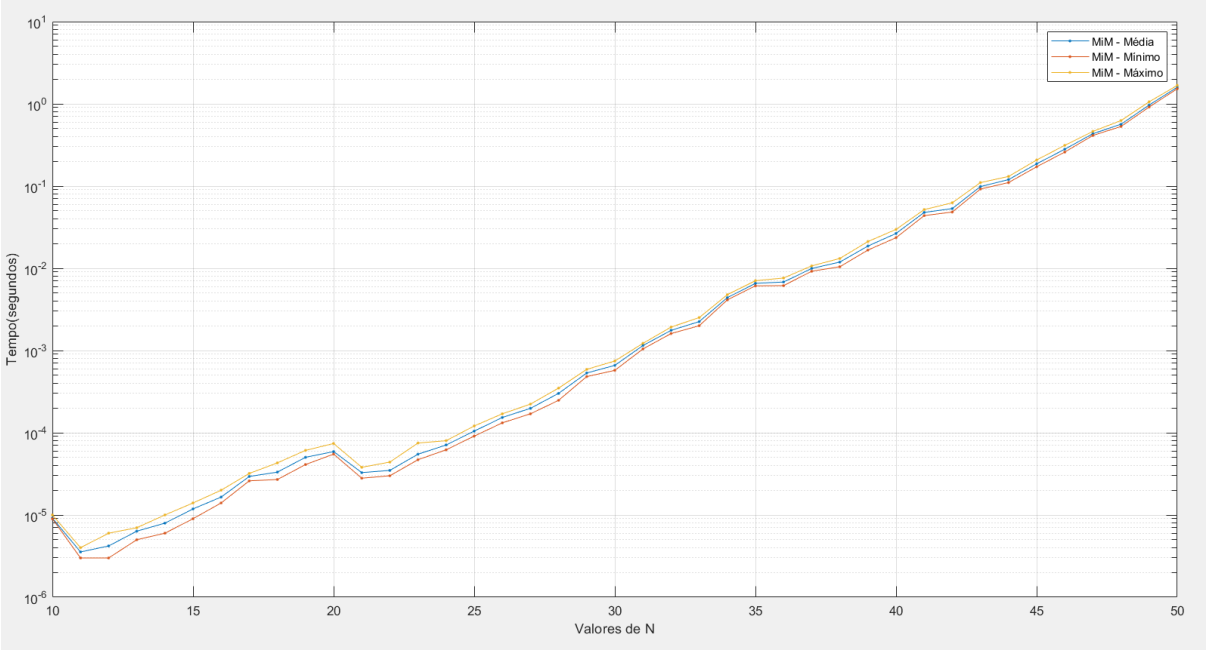
```c
/* low  --> Starting index,  high  --> Ending index */
void quickSort(integer_t array[], integer_t low, integer_t high){

    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        integer_t pi = partition(array, low, high);

        quickSort(array, low, pi - 1);  // Before pi
        quickSort(array, pi + 1, high); // After pi
    }
}


int decimalToBinary(integer_t n, integer_t c, int index, int result[]){


  for (integer_t i = index; i < (c+index); i++)
  {
    integer_t k = (n>>(i-index));

    if (k & 1)
      result[i]=1;
    else
      result[i]=0;
  }

  return 0;
}
```

```c
for (int i = 0; i < n_problems; i++) {
    integer_t *p = all_subset_sum_problems[i].p; // The weights
    int n = all_subset_sum_problems[i].n; // The value of n


       //Tirar proxima linha de comentário se pretende executar o Meet in the Middle bem como a linha 343

       double tempoSums1 = cpu_time();
      int x1 = n/2 + n%2; //tamanho array 1
      int x2 = n/2; //tamanho array 2
      int x3 = 0 ; //counter para adicionar ao segundo array

      integer_t s1 = 1<<x1; //tamanho array soma 1
      integer_t s2 = 1<<x2; //tamanho array soma 2


      integer_t firstHalf[x1]; //criaçao primeiro array
      integer_t secondHalf[x2]; //criaçao segundo array
      integer_t *sum1; //criaçao array soma 1
      integer_t *sum2; //criaçao array soma 2
      integer_t *sum11; //criaçao outro array sum1 usado para o res em binario
      integer_t *sum22; //criaçao outro array sum2 usado para o res em binario
```

```
sum1 = (integer_t *)malloc(s1 * sizeof(integer_t));
sum2 = (integer_t *)malloc(s2 * sizeof(integer_t));
sum11 = (integer_t *)malloc(s2 * sizeof(integer_t));
sum22 = (integer_t *)malloc(s2 * sizeof(integer_t));


//Adicionar os numeros aos arrays 1 e 2
for (int i = 0; i < n; i++){
    if (i < (n/2+n%2)){
        firstHalf[i]=p[i]; //adicionar ao primeiro array
    } else {
        secondHalf[x3]=p[i]; //adicionar ao segundo array
        x3++;
    }
}

//Fazer soma array 1 e adicionar ao array sum1
for (integer_t i = 0;i < s1; i++){

   integer_t soma1=0;

    for (int j = 0; j < n; j++){
        if (i & (1ull<<j)){
            soma1 += firstHalf[j];
        }
    }
    sum1[i] = soma1;
    sum11[i] = soma1;
}
```

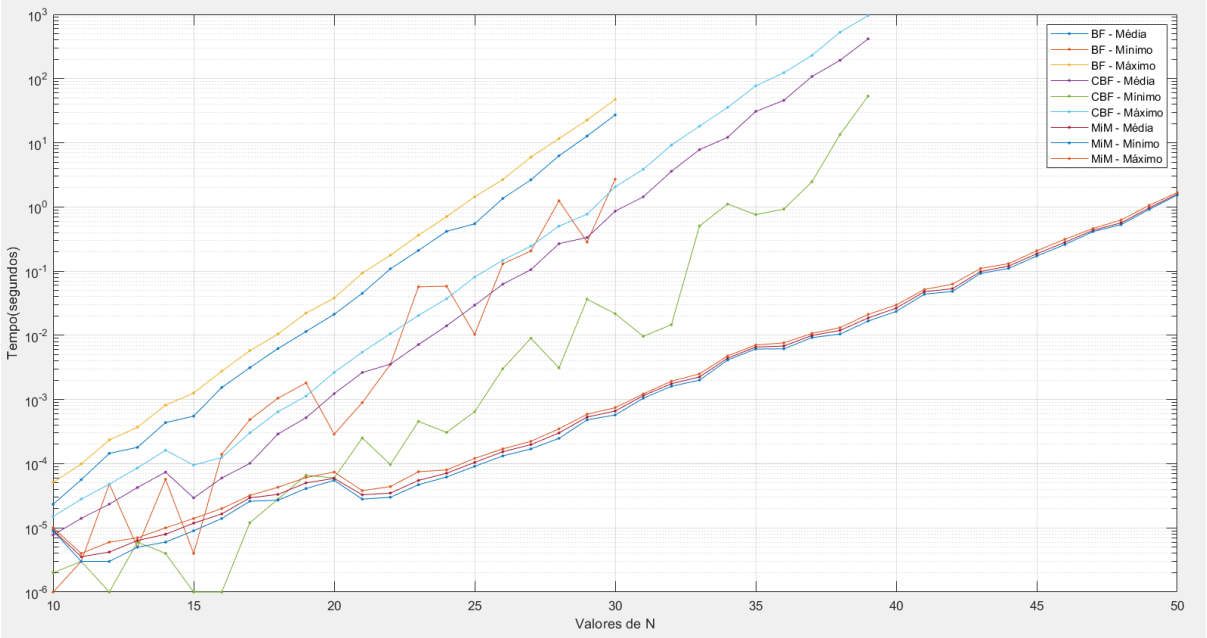
```
//Fazer soma array 2 e adicionar ao array sum2
for (integer_t i = 0; i < s2; i++){

    integer_t soma2=0;

    for (int j = 0; j < n; j++){
        if (i & (1ull<<j)){
            soma2 += secondHalf[j];
        }
    }
    sum2[i] = soma2;
    sum22[i] = soma2;
}

// for (int counter = 0; counter < s1; counter++){
//     printf("%lld  ",sum1[counter]);
// }

quickSort(sum1, 0, s1-1);
quickSort(sum2,0,s2-1);

double tempoSums2 = cpu_time();

double tempoSums = (tempoSums2 - tempoSums1) / (double)20;
```

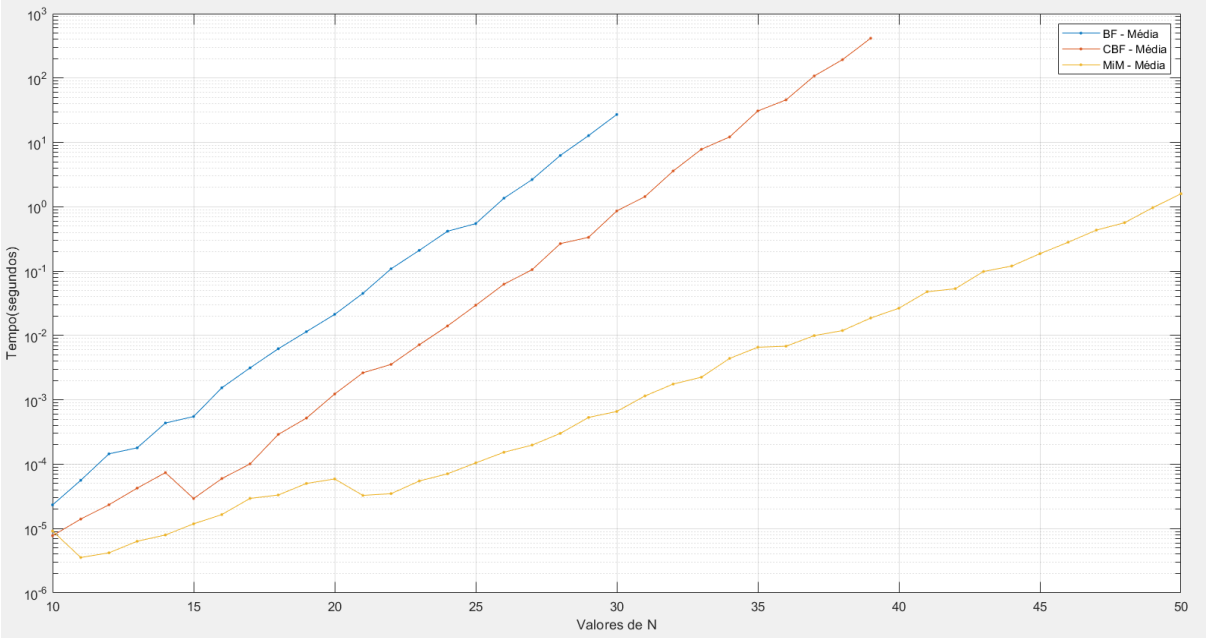
```
found = meetInMiddle(sum1, sum2, s1, s2, desired_sum, result, sum11, sum22, x1, x2);
```




```
Para n = 45 | Found: 1 | Time: 0.290443 seconds Result: 000111000010010011111101010011101001111110101
Para n = 45 | Found: 1 | Time: 0.317297 seconds Result: 000000000000000000000000011000010100110100l101
Para n = 45 | Found: 1 | Time: 0.324090 seconds Result: 000000000000000000000001001000100110011000111
Para n = 45 | Found: 1 | Time: 0.293605 seconds Result: 000000000000000000000001101010101001011001011
Para n = 45 | Found: 1 | Time: 0.288628 seconds Result: 000000000000000000000000111110101010110011101
Para n = 45 | Found: 1 | Time: 0.315288 seconds Result: 000000000000000000000010011111100000011000101
Para n = 45 | Found: 1 | Time: 0.309110 seconds Result: 000000000000000000000001110110110001111011000
Para n = 45 | Found: 1 | Time: 0.296299 seconds Result: 000000000000000000000001100001110011000110001
Para n = 46 | Found: 1 | Time: 0.392660 seconds Result: 100001100011110010001010110100110101101l010111
Para n = 46 | Found: 1 | Time: 0.425246 seconds Result: 011001010101011111101010110101001101001l001111
Para n = 46 | Found: 1 | Time: 0.399222 seconds Result: 011000011011011010000011001110000101101110011O110
Para n = 46 | Found: 1 | Time: 0.405243 seconds Result: 000000010011100011011110110111011101111010l0101
Para n = 46 | Found: 1 | Time: 0.399855 seconds Result: 001100101010000100001111100110111000000101001O
Para n = 46 | Found: 1 | Time: 0.402119 seconds Result: 000101010111111111111001100110011010011111l101
Para n = 46 | Found: 1 | Time: 0.426013 seconds Result: 000000111111100101101011100011100011010110001O
Para n = 46 | Found: 1 | Time: 0.430940 seconds Result: 010011101111011110001011000001110001011101001l
Para n = 46 | Found: 1 | Time: 0.393371 seconds Result: 111011111101010000000101011011111011000011000
Para n = 46 | Found: 1 | Time: 0.407729 seconds Result: 001000010111011000010010000111010010000001l001
Para n = 46 | Found: 1 | Time: 0.381364 seconds Result: 001000011001000110000001011000011100000010l100
Para n = 46 | Found: 1 | Time: 0.425087 seconds Result: 010101000001111110010111100100110010100111000
Para n = 46 | Found: 1 | Time: 0.423881 seconds Result: 011110011110111011001010110001111100111100l010
Para n = 46 | Found: 1 | Time: 0.442167 seconds Result: 000110111101111100101110100011001000111010000l
Para n = 46 | Found: 1 | Time: 0.388406 seconds Result: 000011010100100000100100111011000100110111100l1
Para n = 46 | Found: 1 | Time: 0.410379 seconds Result: 101101000001111001001101001000000010000l010011
Para n = 46 | Found: 1 | Time: 0.384946 seconds Result: 010010100000000100101000000100110001000011l000
Para n = 46 | Found: 1 | Time: 0.414310 seconds Result: 011111000101101100101110111000010111111001O111
Para n = 46 | Found: 1 | Time: 0.410448 seconds Result: 101111111001010001111011010001100111101110l101
Para n = 46 | Found: 1 | Time: 0.430277 seconds Result: 111110100000110010111110011111100000001011O111
Para n = 47 | Found: 1 | Time: 0.947184 seconds Result: 000000000000000000000000011000100100000001101001
Para n = 47 | Found: 1 | Time: 0.994207 seconds Result: 000000000000000000000000001110110111011000001
Para n = 47 | Found: 1 | Time: 0.944422 seconds Result: 000000000000000000000000100111010000000011001O0
Para n = 47 | Found: 1 | Time: 0.960051 seconds Result: 110010100011110011100110100010000001000010100l1
Para n = 47 | Found: 1 | Time: 0.962280 seconds Result: 001100101100101111101110011000111011111101101l
Para n = 47 | Found: 1 | Time: 0.971922 seconds Result: 000000000000000000000000011111100110110101l101
Para n = 47 | Found: 1 | Time: 0.974286 seconds Result: 000000000000000000000000011010000111100110000l
Para n = 47 | Found: 1 | Time: 0.986525 seconds Result: 000000000000000000000000110110111101010010001l1
Para n = 47 | Found: 1 | Time: 0.950210 seconds Result: 000000000000000000000000101111000100010001111l1
Para n = 47 | Found: 1 | Time: 0.928408 seconds Result: 110111101101110100100100101110111001011101111l1
Para n = 47 | Found: 1 | Time: 0.957096 seconds Result: 000000000000000000000000010010000100001010000l1
Para n = 47 | Found: 1 | Time: 0.972613 seconds Result: 000000000000000000000000011110010000111101001O1
Para n = 47 | Found: 1 | Time: 0.966644 seconds Result: 000000000000000000000000100101000010010110010l1
Para n = 47 | Found: 1 | Time: 0.970796 seconds Result: 000111010110101010100110000101010000110001001O0
Para n = 47 | Found: 1 | Time: 0.966678 seconds Result: 000000000000000000000000110010001010101000111l1
Para n = 47 | Found: 1 | Time: 1.004296 seconds Result: 000000000000000000000000101101010010000110000l0
Para n = 47 | Found: 1 | Time: 0.946980 seconds Result: 000000000000000000000000000010011111110101111110
Para n = 47 | Found: 1 | Time: 0.961359 seconds Result: 000000000000000000000000100111001001001100100001
Para n = 47 | Found: 1 | Time: 0.961168 seconds Result: 000000000000000000000000011010100101110011100
Para n = 47 | Found: 1 | Time: 0.980748 seconds Result: 000000000000000000000000100000100101011100000l1
```

Gráficos de comparação entre os diferentes métodos





Código do programa em Matlab

```
Matriz = readmatrix("results");

temposF0 = reshape(Matriz(Matriz(:, 1) == 0, 5), 20, []);
averageF0 = mean(temposF0);
minF0 = min(temposF0);
maxF0 = max(temposF0);

temposF1 = reshape(Matriz(Matriz(:, 1) == 1, 5), 20, []);
averageF1 = mean(temposF1);
minF1 = min(temposF1);
maxF1 = max(temposF1);

temposF2 = reshape(Matriz(Matriz(:, 1) == 2, 5), 20, []);
averageF2 = mean(temposF2);
minF2 = min(temposF2);
maxF2 = max(temposF2);

temposF3 = reshape(Matriz(Matriz(:, 1) == 3, 5), 20, []);
averageM3 = mean(temposF3);
minF3 = min(temposF3);
maxF3 = max(temposF3);


figure(1);
semilogy((1:length(averageF0)) + 9, averageF0, ".-", 'DisplayName', "BF - Média"); hold on;
semilogy((1:length(minF0)) + 9, minF0, ".-", 'DisplayName', "BF - Mínimo"); hold on;
semilogy((1:length(maxF0)) + 9, maxF0, ".-", 'DisplayName', "BF - Máximo"); hold on;
xlabel("Valores de N");
ylabel("Tempo(segundos)");
grid on;
legend;
```

```
figure(2);
semilogy((1:length(averageF1)) + 9, averageF1, ".-", 'DisplayName', "CBF - Média"); hold on;
semilogy((1:length(minF1)) + 9, minF1, ".-", 'DisplayName', "CBF - Mínimo"); hold on;
semilogy((1:length(maxF1)) + 9, maxF1, ".-", 'DisplayName', "CBF - Máximo"); hold on;
xlabel("Valores de N");
ylabel("Tempo(segundos)");
grid on;
legend;

figure(3);
semilogy((1:length(averageF2)) + 9, averageF2, ".-", 'DisplayName', "MiM - Média"); hold on;
semilogy((1:length(minF2)) + 9, minF2, ".-", 'DisplayName', "MiM - Mínimo"); hold on;
semilogy((1:length(maxF2)) + 9, maxF2, ".-", 'DisplayName', "MiM - Máximo"); hold on;
xlabel("Valores de N");
ylabel("Tempo(segundos)");
grid on;
legend;



figure(4);
semilogy((1:length(averageF0)) + 9, averageF0, ".-", 'DisplayName', "BF - Média"); hold on;
semilogy((1:length(averageF1)) + 9, averageF1, ".-", 'DisplayName', "CBF - Média"); hold on;
semilogy((1:length(averageF2)) + 9, averageF2, ".-", 'DisplayName', "MiM - Média"); hold on;
xlabel("Valores de N");
ylabel("Tempo(segundos)");
grid on;
legend;
```

```
figure(5);
semilogy((1:length(averageF0)) + 9, averageF0, ".-", 'DisplayName', "BF - Média"); hold on;
semilogy((1:length(minF0)) + 9, minF0, ".-", 'DisplayName', "BF - Mínimo"); hold on;
semilogy((1:length(maxF0)) + 9, maxF0, ".-", 'DisplayName', "BF - Máximo"); hold on;


semilogy((1:length(averageF1)) + 9, averageF1, ".-", 'DisplayName', "CBF - Média"); hold on;
semilogy((1:length(minF1)) + 9, minF1, ".-", 'DisplayName', "CBF - Mínimo"); hold on;
semilogy((1:length(maxF1)) + 9, maxF1, ".-", 'DisplayName', "CBF - Máximo"); hold on;

semilogy((1:length(averageF2)) + 9, averageF2, ".-", 'DisplayName', "MiM - Média"); hold on;
semilogy((1:length(minF2)) + 9, minF2, ".-", 'DisplayName', "MiM - Mínimo"); hold on;
semilogy((1:length(maxF2)) + 9, maxF2, ".-", 'DisplayName', "MiM - Máximo"); hold on;

xlabel("Valores de N");
ylabel("Tempo(segundos)");
grid on;
legend;
```

Conclusão

Da realização deste trabalho conseguimos concluir que para qualquer tipo de problema, existem diferentes soluções, cada uma com diferentes complexidades computacionais e diferentes tempos de resolução.
Também conseguimos concluir que é sempre possível melhorar o código e tornar este mais rápido e eficaz.

Bibliografia

https://www.cyberciti.biz/faq/bash-scripting-using-awk/

https://docs.oracle.com/cd/E19504-01/802-5753/6i9g71m3i/index.html

https://stackoverflow.com/questions/16483119/an-example-of-how-to-use-getopts-in-bash

https://www.geeksforgeeks.org/quick-sort/

https://www.javatpoint.com/c-program-to-convert-decimal-to-binary